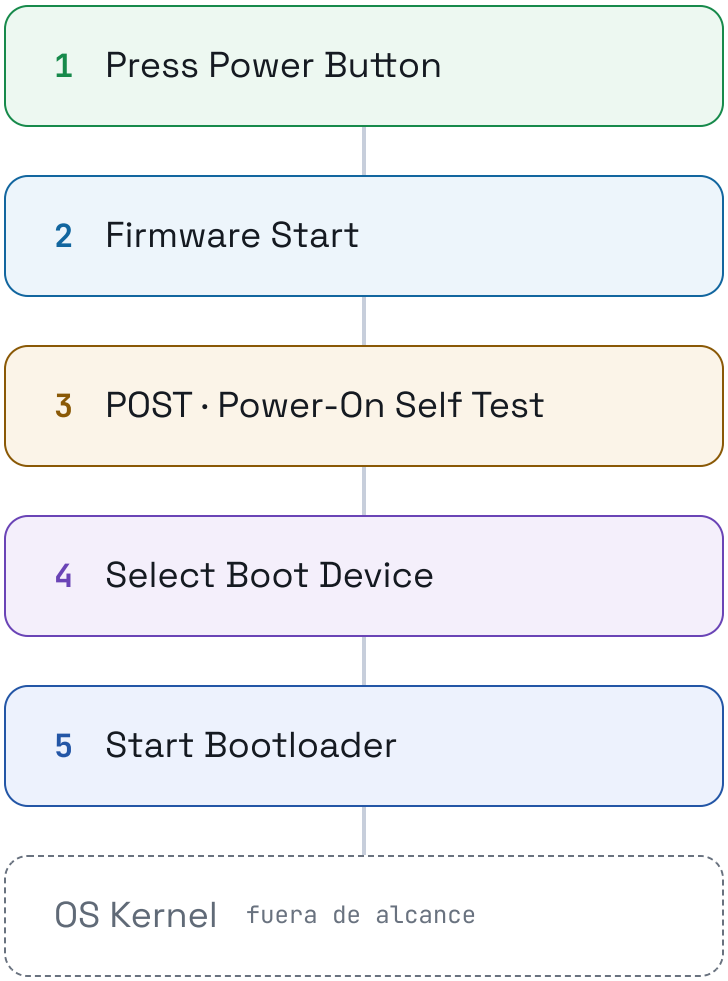
DOCUMENTO · REFERENCIA TÉCNICA

Guía Exhaustiva del Boot Process

Complemento del video · @sbimoxa

Alcance. Arquitectura PC estándar x86 / x86-64 con firmware BIOS (legacy) o UEFI. Los sistemas embebidos, el boot ROM de SoC ARM o los smartphones siguen lógicas distintas y no están cubiertos aquí.

DIAGRAMA GENERAL



FASE 1

Press Power Button

1.1 El circuito físico

El power button no enciende la fuente de poder (**PSU**, Power Supply Unit) directamente. Está conectado a un header de 2 pines en la motherboard (`PWR_SW` / `POWER_SW`, parte del front panel connector) y funciona como un simple switch momentáneo que la circuitería de la motherboard monitorea constantemente.

1.2 Standby power (5VSB)

Mientras la PSU esté conectada a corriente alterna y su switch trasero esté en ON, existe un riel de standby power de +5V (**5VSB**) que alimenta permanentemente ciertos componentes: el chipset / Super I/O, el circuito que detecta el power button, controladores de red con Wake-on-LAN, etc. Esto es lo que permite que el equipo "escuche" el botón aun estando apagado.

1.3 Secuencia ATX de soft power-on

1. Se detecta la pulsación (con **debouncing** por hardware/firmware para filtrar rebotes eléctricos).
2. El chipset / Embedded Controller (EC) activa la señal **`PS_ON#`** (activa en bajo) hacia la PSU a través del conector ATX principal.
3. La PSU energiza todos sus rieles: +3.3V, +5V, +12V (y −12V en fuentes más antiguas).
4. La PSU verifica internamente que los voltajes estén dentro de tolerancia.
5. Una vez estables, la PSU activa la señal **Power Good** (`PWR_OK` / `PG`) hacia la motherboard.
6. El chipset mantiene la línea **`RESET#`** del CPU activa hasta recibir Power Good; solo entonces la libera.

7 Con RESET# liberado, el CPU puede empezar a hacer fetch de instrucciones. Aquí termina la Fase 1.

1.4 Notas adicionales

- **Sistemas AT (pre-ATX):** el switch cortaba la corriente AC mecánicamente. No existía soft power-on ni standby power.
- **Laptops:** el Embedded Controller gestiona la secuencia considerando también batería y cargador, pero el concepto de standby power + señal de "todo estable" hacia el resto del sistema es análogo.
- **Wake events alternativos:** el mismo standby power habilita otros disparadores sin tocar el botón físico: Wake-on-LAN, RTC alarm, USB wake.

FASE 2 Firmware Start

2.1 El Reset Vector

Al liberarse RESET#, el CPU x86 arranca siempre en **Real Mode** con estado arquitectónico fijo: CS = `F000h` con base oculta `FFFF0000h` (no el cálculo normal de real mode), IP = `FFF0h`. La dirección física efectiva resultante es:

```
0xFFFF0000 + 0xFFF0 = 0xFFFFFFF0
```

Esa dirección (16 bytes por debajo de los 4 GB) es el **reset vector**. El chipset redirige accesos a esa región al chip de **SPI Flash** donde vive el firmware, no a RAM (todavía no inicializada).

2.2 Execute-in-place (XIP)

Sin RAM utilizable, el firmware ejecuta directamente desde la SPI Flash (**XIP**). Como en el reset vector solo caben 16 bytes, ahí normalmente hay un `JMP FAR` hacia el código real de inicialización (a veces llamado *bootblock*).

2.3 Microcode update

El firmware suele cargar un **microcode update**: parches binarios de Intel/AMD que corrigen errata de silicio, cargados en SRAM interna del CPU. Esto se repite en cada boot porque el parche no persiste tras un power cycle.

2.4 Cache-As-RAM (CAR)

Sin RAM inicializada no hay stack, así que no se puede ejecutar código en C, solo assembly puro. La solución estándar es **Cache-As-RAM (CAR)**: se configura la cache L2/L3 del CPU en un modo especial para que actúe como RAM temporal, habilitando un stack y código de más alto nivel.

2.5 Memory initialization (MRC / memory training)

Uno de los pasos más largos del firmware start:

- Se lee la **SPD** (Serial Presence Detect): una EEPROM en cada DIMM con capacidad, timings (CL, tRCD, tRP, tRAS), voltaje y fabricante.
- El **Memory Reference Code (MRC)** en Intel (o AGESA en AMD) configura el controlador de memoria, calibra impedancias y delays por pin (*memory training*), y valida que el canal funcione establemente.
- Puede tomar de cientos de milisegundos a varios segundos, según capacidad, cantidad de módulos, y si el firmware tiene cacheado un training previo.
- Al terminar, el firmware transiciona de Cache-As-RAM a RAM real, copiándose a sí mismo a DRAM para ejecutar más rápido de ahí en adelante.

2.6 Fases formales de UEFI (PI Spec)

La especificación **UEFI Platform Initialization (PI)** define fases formales que mapean a lo anterior:

FASE UEFI PI	QUÉ HACE
SEC Security	Código mínimo en assembly; establece Cache-As-RAM; primer punto de confianza (root of trust en sistemas con Boot Guard / PSB).
PEI Pre-EFI Init	Inicializa CPU, chipset y memoria; produce **HOBs** (Hand-Off Blocks) describiendo el estado del sistema.
DXE Driver Exec Env	La mayor parte de la inicialización real: decenas/cientos de DXE drivers inicializan PCIe, USB, storage, video, etc. Expone **Boot Services** y **Runtime Services**.
BDS Boot Device Sel	Corresponde a la Fase 4 de esta guía.
TSL Transient Sys Load	Donde corre el bootloader / OS loader (Fase 5), todavía con Boot Services disponibles.

En **legacy BIOS** no existe esta separación formal: es un flujo más monolítico de rutinas en 16-bit real mode.

2.7 Enumeración de dispositivos e inicialización de video

Durante DXE (o su equivalente en legacy BIOS), el firmware:

- Enumera el bus **PCIe**: detecta dispositivos, asigna recursos (direcciones MMIO, IRQs), entrena los enlaces (negocia ancho/velocidad).
- Inicializa controladores **USB** (necesario para que un teclado USB funcione en el setup del firmware).

- Inicializa video: en legacy, vía el **Video BIOS / Option ROM** de la GPU (código x86 de 16 bits embebido en la tarjeta); en UEFI, vía el protocolo **GOP** (Graphics Output Protocol). Aquí aparece el logo del fabricante en pantalla.
- Genera tablas de descripción de sistema que el OS usará después: **ACPI** (`MADT`, `DSDT`, `FADT`, entre otras) y **SMBIOS**.

2.8 BIOS legacy vs UEFI — diferencia de fondo

	LEGACY BIOS	UEFI
Modo de CPU	16-bit real mode	32/64-bit, flat memory + paging
Arquitectura	Monolítica, interrupt-based (INT 13h, INT 10h…)	Modular, basada en drivers (DXE) y protocolos
Partición de disco	MBR	GPT (puede leer MBR también)
Tamaño de firmware	Cientos de KB	Varios MB
Red en firmware	No (salvo PXE Option ROM)	Sí, stack de red nativo
Seguridad	Sin verificación de firma	Secure Boot

FASE 3

POST · Power-On Self Test

3.1 Qué es

Secuencia de diagnóstico que verifica que el hardware esencial esté presente y funcional antes de intentar arrancar un OS. En BIOS legacy era una fase bien delimitada; en UEFI está entrelazada con la ejecución de los DXE drivers (cada driver valida el hardware que inicializa), pero el concepto persiste.

3.2 Qué verifica típicamente

- Registros y funcionalidad básica del CPU.
- Checksum de la **CMOS** (memoria de configuración respaldada por batería, donde vive el RTC y las settings del firmware).
- Timers / contadores del sistema.
- Controladores de interrupciones (**PIC / APIC**) y DMA.
- Presencia y conteo básico de RAM (los sistemas modernos ya no hacen un test exhaustivo por defecto; el memory training de la Fase 2 ya validó el canal).
- Presencia de teclado / controlador de teclado.
- Inicialización del adaptador de video.
- Detección de controladores de almacenamiento y dispositivos conectados.

3.3 Reporte de errores

Cuando el fallo ocurre **antes** de que el video esté disponible, el firmware no puede mostrar texto, así que usa dos mecanismos alternativos:

- **Beep codes:** patrones de pitidos por el speaker de la motherboard. El significado depende del fabricante del firmware (AMI, Award/Phoenix, etc. usan códigos distintos entre sí); no hay estándar único, conviene consultar el manual específico en vez de asumir un patrón universal.
- **POST codes:** códigos numéricos en hexadecimal escritos al puerto de I/O `0x80`, leíbles con una tarjeta debug / POST card o mostrados en un display de 7 segmentos integrado (común en motherboards de gama alta).

Una vez inicializado el video, los errores subsiguientes pueden mostrarse como texto en pantalla.

3.4 Fast Boot

Muchos firmwares UEFI modernos ofrecen **Fast Boot / Ultra Fast Boot**, que omite deliberadamente ciertos pasos (inicialización de USB legacy, chequeos redundantes) para reducir el tiempo de arranque. Trade-off: con Ultra Fast Boot activado puede no ser posible interrumpir el arranque para entrar al setup sin un método alternativo (botón físico dedicado o forzar reinicio desde el OS hacia el firmware).

3.5 Fin de la fase

Si un componente crítico falla (CPU no detectado, sin RAM, checksum de CMOS irrecuperable), el arranque se detiene ahí (**halt**) con el error reportado vía beep / POST code. Si el POST pasa —incluso con warnings no críticos— el firmware continúa a la Fase 4.

FASE 4

Select Boot Device

4.1 Mecanismo Legacy BIOS

- El firmware mantiene una **boot order** configurada (guardada en CMOS/NVRAM): p. ej. HDD1, USB, CD/DVD, red.
- Para cada dispositivo en ese orden, lee su primer sector (512 bytes): el **MBR** (Master Boot Record).
- Verifica la **boot signature**: los últimos 2 bytes del sector deben ser `0x55` y `0xAA` (en ese orden en disco), lo que se conoce como signature `0xAA55` al interpretar esos bytes como palabra little-endian. Si coincide, el dispositivo se considera bootable.
- Si es válido, el firmware carga ese sector completo en RAM en la dirección física `0x7C00` y transfiere ejecución ahí. Ese es, literalmente, el primer código del bootloader (Fase 5).
- Si no coincide la firma, pasa al siguiente dispositivo.

4.2 Mecanismo UEFI

Sustancialmente más sofisticado:

- El firmware mantiene variables en **NVRAM**: `BootOrder` (lista de prioridad) y entradas individuales `Boot0000`, `Boot0001`, etc., cada una describiendo un boot option (device path + descripción + argumentos). Lo gestiona el **Boot Manager** (parte de BDS).
- Busca una **EFI System Partition (ESP)**: partición formateada en FAT32, identificada por un GUID de tipo de partición específico en la tabla **GPT**.

- Dentro de la ESP busca el ejecutable indicado por la entrada NVRAM correspondiente. Rutas típicas:

```
Windows  \EFI\Microsoft\Boot\bootmgfw.efi
Linux    \EFI\<distro>\shimx64.efi → grubx64.efi
fallback \EFI\BOOT\BOOTX64.EFI  (o BOOTIA32/BOOTAA64)
```

- Valida que el archivo sea un ejecutable **PE32+** válido con subsistema de aplicación EFI.

4.3 Secure Boot

Con Secure Boot habilitado, el firmware verifica la firma digital del `.efi` contra una jerarquía de llaves almacenada en el propio firmware:

- **PK** (Platform Key): llave raíz, controlada por el fabricante / OEM.
- **KEK** (Key Exchange Key): autoriza actualizaciones de la signature database.
- **db** (Signature Database): certificados / hashes de bootloaders autorizados.
- **dbx** (Forbidden Signature Database): certificados / hashes revocados.

Un binario no firmado, firmado con llave no confiable, o presente en `dbx`, es rechazado.

Relacionado pero distinto: **Measured Boot**, donde un **TPM** (Trusted Platform Module) calcula hashes de cada componente del boot chain y los almacena en registros **PCR** (Platform Configuration Registers). No bloquea el arranque como Secure Boot, pero habilita *attestation* posterior (BitLocker usa PCRs para decidir si libera la clave de cifrado).

4.4 Interacción del usuario

- **Boot Menu temporal (one-time):** una tecla en el POST (`F12`, `F8`, `Esc`, según fabricante) permite elegir manualmente un dispositivo distinto sin alterar la configuración persistente.

- **Firmware Setup:** otra tecla (`F2`, `Del`, según fabricante) da acceso a la configuración persistente, incluido el boot order.

4.5 Arranque por red (PXE)

Si está habilitado y priorizado, el firmware / Option ROM de la NIC ejecuta **PXE** (Preboot Execution Environment): envía una solicitud DHCP para obtener IP y la dirección de un servidor de arranque, y descarga una imagen de arranque vía TFTP.

FASE 5 Start Bootloader

5.1 Cadena Legacy BIOS (MBR)

El MBR completo son 512 bytes: **446 bytes** de código ejecutable + **64 bytes** de tabla de particiones (4 entradas × 16 bytes) + **2 bytes** de firma. Con solo 446 bytes, el código del MBR no puede entender sistemas de archivos, así que el arranque se hace en etapas (usando GRUB, el bootloader más común en Linux, como ejemplo):

1. **Stage 1** (el MBR mismo): busca la partición marcada como *activa* y/o carga el siguiente stage desde sectores fijos.
2. **Stage 1.5:** escrito en el "MBR gap" (sectores entre el MBR y la primera partición). Contiene drivers mínimos de filesystem, permitiendo leer Stage 2 desde una partición formateada en vez de sectores crudos.
3. **Stage 2:** el bootloader completo (`core.img` / módulos de GRUB). Lee `grub.cfg`, puede mostrar un menú (selección de OS/kernel/recovery), y carga el kernel elegido en RAM.

En Windows legacy (pre-UEFI), el equivalente es MBR → Boot Sector de la partición → **Boot Manager** (`bootmgr`), que lee el **BCD** (Boot Configuration Data) y carga `winload.exe`.

5.2 Cadena UEFI

Más directa, porque UEFI ya sabe leer FAT32 y ejecutar binarios PE32+ nativamente, sin MBR intermedio:

- El firmware carga el `.efi` seleccionado en la Fase 4 directamente en RAM y transfiere control.
- **Windows:** `bootmgfw.efi` (Windows Boot Manager) lee la BCD, decide qué instalación arrancar (menú si hay varias) y carga `winload.efi`, que carga el kernel (`ntoskrnl.exe`), el HAL y los drivers boot-critical.
- **Linux con GRUB:** `shimx64.efi` (firmado por una CA confiable para Secure Boot) valida y encadena a `grubx64.efi`, que lee `grub.cfg` y carga `vmlinuz` e `initramfs` en memoria.
- **Alternativa más directa:** el kernel puede compilarse con **EFI stub**, permitiendo que UEFI lo ejecute directamente como aplicación EFI sin GRUB de por medio (usado, por ejemplo, por `systemd-boot`).

5.3 Qué hace el bootloader antes de entregar el control

Independientemente de BIOS/UEFI, el bootloader típicamente:

- Construye/obtiene un **mapa de memoria** para que el kernel sepa qué regiones de RAM puede usar y cuáles están reservadas (firmware, ACPI, MMIO). En legacy, vía `INT 15h, AX=E820h` (el "E820 map"); en UEFI, vía `GetMemoryMap()`.
- Prepara y pasa **parámetros de arranque / command line** al kernel (p. ej. ubicación del root filesystem, opciones de arranque).
- Carga el **initrd / initramfs** (Linux) o equivalente: filesystem temporal en RAM con los drivers mínimos para montar el root filesystem real (RAID, LVM, cifrado de disco, NVMe si no está compilado en el kernel).

- Realiza la transición final de modo de CPU si es necesario: en legacy, real mode → protected mode → long mode, incluyendo históricamente habilitar la **A20 line** para direccionar memoria por encima de 1 MB.
- En UEFI, el bootloader (o el propio kernel vía EFI stub) llama a `ExitBootServices()`: el punto de no retorno donde el firmware deja de tener control activo y el OS toma posesión completa del hardware.
- Finalmente, salta al entry point del kernel.

> En ese punto termina, estrictamente, el boot process de firmware + bootloader. Lo que sigue (inicialización del kernel, montaje del root filesystem, arranque de un init system como `systemd`) es la etapa de **OS startup**, fuera del alcance de esta guía.

Comparativa final: Legacy BIOS vs UEFI por fase

FASE	LEGACY BIOS	UEFI
1 · Power Button	Puramente eléctrico/mecánico, no depende del firmware.	Igual.
2 · Firmware Start	Flujo monolítico en 16-bit real mode.	SEC → PEI → DXE, modular, 32/64-bit.
3 · POST	Fase relativamente separada y explícita.	Entrelazada con la ejecución de DXE drivers.
4 · Select Boot Device	Boot order + escaneo de MBR (signature 0xAA55).	Boot Manager + variables NVRAM + ESP (FAT32) + Secure Boot.
5 · Start Bootloader	MBR (446B) → Stage 1.5 → Stage 2.	Ejecución directa de .efi (PE32+) desde la ESP.

Glosario

`BIOS`	Basic Input/Output System — firmware legacy, real mode.
`UEFI`	Unified Extensible Firmware Interface — firmware moderno, modular.
`POST`	Power-On Self Test.
`MBR`	Master Boot Record — esquema de partición legacy + primer sector booteable.
`GPT`	GUID Partition Table — esquema de partición moderno usado con UEFI.
`ESP`	EFI System Partition — partición FAT32 donde viven los .efi.
`DXE`	Driver Execution Environment — fase UEFI donde se inicializa la mayoría del hardware.
`CAR`	Cache-As-RAM — uso de cache de CPU como RAM temporal antes de que la memoria esté lista.
`SPD`	Serial Presence Detect — EEPROM en un DIMM con sus specs / timings.
`ACPI`	Advanced Configuration and Power Interface — tablas que describen hardware / power management al OS.
`initrd / initramfs`	Filesystem temporal en RAM usado por Linux en las etapas tempranas de boot.
`TPM`	Trusted Platform Module — chip de seguridad usado para Measured Boot, entre otras cosas.